CHAPTER – THREE

PROCESS MODELS

3.1 Overview

In software engineering, a software development process is the process of dividing software development work into distinct phases to improve design, product management, and project management. It is also known as a software development life cycle. The methodology may include the pre-definition of specific deliverables and artifacts that are created and completed by a project team to develop or maintain an application

Some people consider a life-cycle "model" a more general term for a category of methodologies and a software development "process" a more specific term to refer to a specific process chosen by a specific organization.

3.2 Process Models

A Process Model describes the sequence of phases for the entire lifetime of a product. Therefore, it is sometimes also called Product Life Cycle. This covers everything from the initial commercial idea until the final de-installation or disassembling of the product after its uses.

Usually there are three main phases:

- concept phase
- implementation phase
- maintenance phase

3.2.1 Waterfall Model

The waterfall model is believed to have been the first process model which was introduced and widely followed in software engineering. The innovation was that the first time software engineering was divided into separate phases. Programs were very small, the requirements only a few and after punching a pile of cards the program was done and could be tested by inserting it into the card reader and observing what it did

As programs became bigger the need for a better requirements phase, some more thoughts on the design, etc. were needed. Programmers found it more and more difficult to keep an abstract of the program in their mind and transfer it into code. Also the thought of having a separate testing phase performed by dedicated testers evolved. The different phases of software engineering were identified and simply cascaded in each other, allowing for loops in case it was found in a subsequent phase that the previous phase was not done properly.

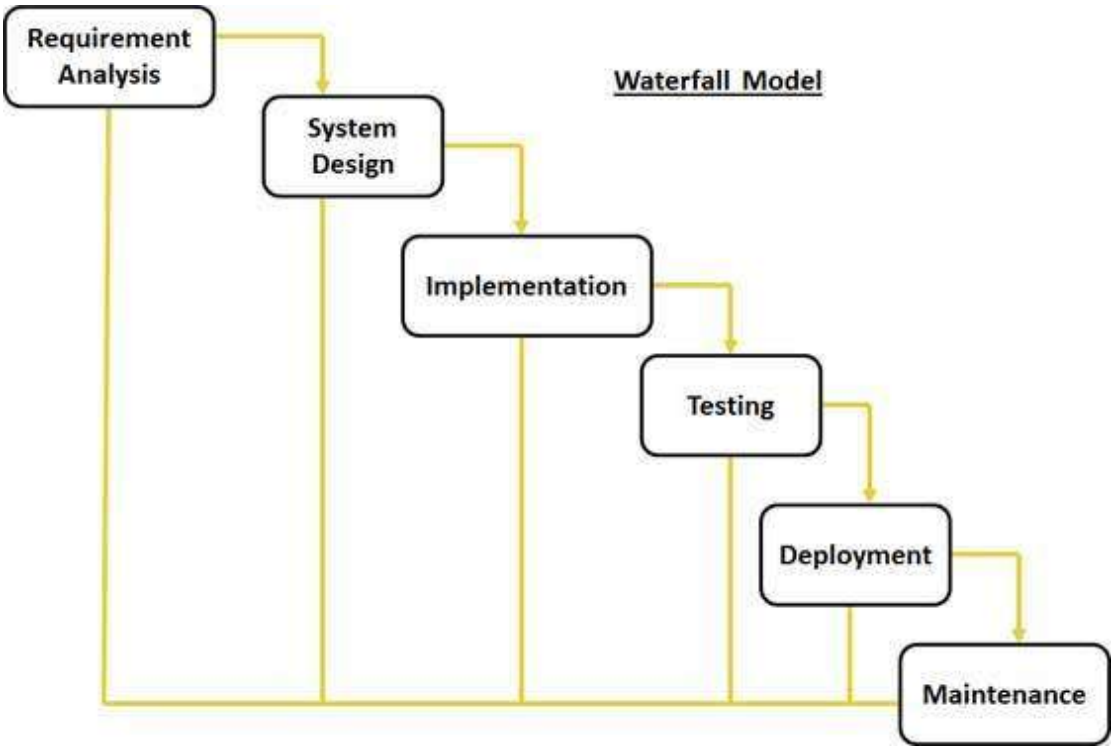

Fig 3.1 Waterfall Model

3.2.2 Spiral Model

The process begins at the centre position. From there it moves clockwise in traversals. Each traversal of the spiral usually results in a deliverable. It is not clearly defined what this deliverable is. This changes from traversal to traversal. For example, the first traversals may result in a requirement specification. The second will result in a prototype, and the next one will result in another prototype or sample of a product, until the last traversal leads to a product which is suitable to be sold. Consequently, the related activities and their documentation will also mature towards the outer traversals. E.g. a formal design and testing session would be placed into the last traversal

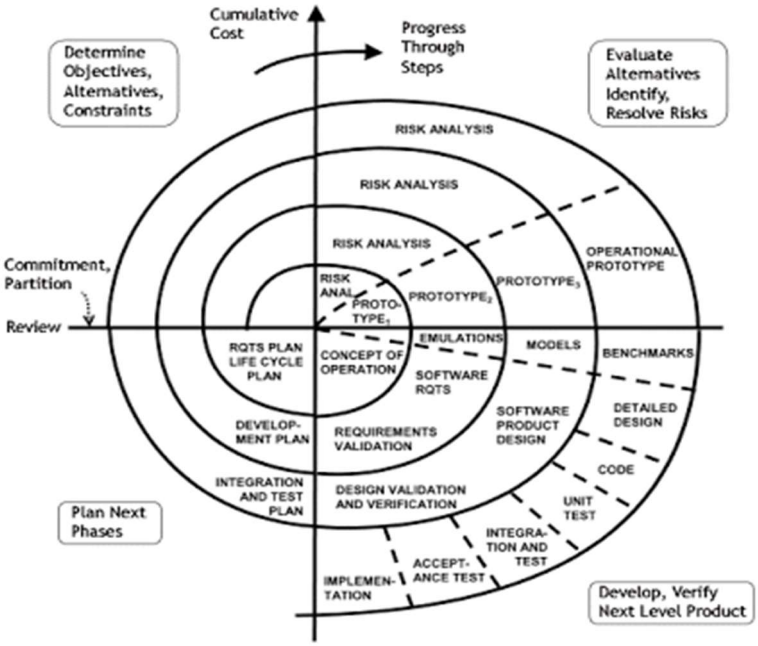

Fig 3.2 Spiral Model

3.2.3 Cocomo Model

COCOMO consists of a hierarchy of three increasingly detailed and accurate forms. The first level, Basic COCOMO is good for quick, early, rough order of magnitude estimates of software costs, but its accuracy is limited due to its lack of factors to account for difference in project attributes (Cost Drivers). Intermediate COCOMO takes these Cost Drivers into account and Detailed COCOMO additionally accounts for the influence of individual project phases. Basic COCOMO computes software development effort (and cost) as a function of program size. Program size is expressed in estimated thousands of lines of code (KLOC).

3.2.4 Agile Model

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds. These builds are provided in iterations. Each iteration typically lasts from about one to three weeks. Every iteration involves cross functional teams working simultaneously on various areas like −

- Planning
- Requirements Analysis

- Design
- Coding
- Unit Testing and
- Acceptance Testing.

At the end of the iteration, a working product is displayed to the customer and important stakeholders.

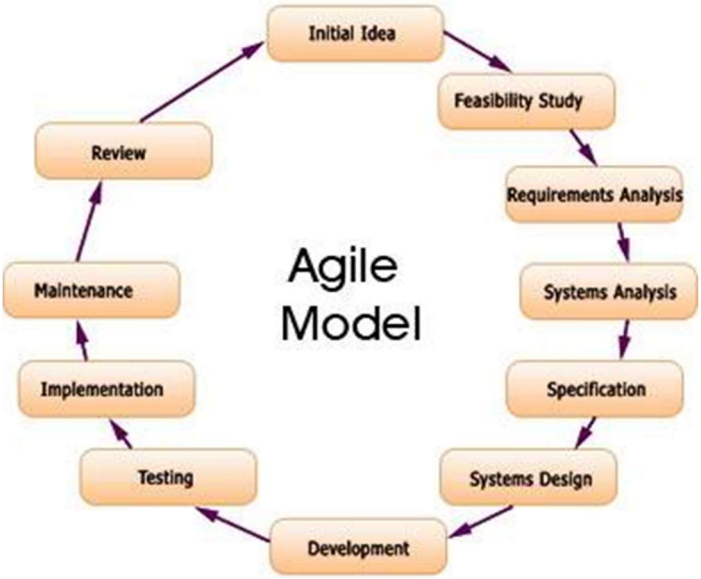

Fig. 3.3 Agile Model

3.3 Proposed Model

The proposed model for this project is prototype model.

A prototype is a toy implementation of the system. A prototype usually exhibits limited functional capabilities, low reliability, and inefficient performance compared to the actual software. A prototype is usually built using several shortcuts. The shortcuts might involve using inefficient, inaccurate, or dummy functions. The shortcut implementation of a function, for example, may produce the desired results by using a table look-up instead of performing the actual computations. A prototype usually turns out to be a very crude version of the actual system.

Need for a prototype in software development

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs:

• How the screens might look like

• How the user interface would behave

• How the system would produce outputs

This is something similar to what the architectural designers of a building do; they show a prototype of the building to their customer. The customer can evaluate whether he likes it or not and the changes that he would need in the actual product. A similar thing happens in the case of a software product and its prototyping model.

Another reason for developing a prototype is that it is impossible to get the perfect product in the first attempt. Many researchers and engineers advocate that if you want to develop a good product you must plan to throw away the first version. The experience gained in developing the prototype can be used to develop the final product.

A prototyping model can be used when technical solutions are unclear to the development team. A developed prototype can help engineers to critically examine the technical issues associated with the product development. Often, major design decisions depend on issues like the response time of a hardware controller, or the efficiency of a sorting algorithm, etc. In such circumstances, a prototype may be the best or the only way to resolve the technical issues.

Examples for prototype model

A prototype of the actual product is preferred in situations such as:

• User requirements are not complete

• Technical issues are not clear

A prototype is made first and final product is developed based on it. A prototype is a model or a program which is not based on strict planning, but is an early approximation of the final product or software system. A prototype acts as a sample to test the process. From this sample, we learn and try to build a better final product. Please note that this prototype may or may not be completely different from the final system we are trying to develop. This type of System Development Method is employed when it is very difficult to obtain exact requirements from the customer. While making the model, user keeps giving feedbacks from time to time and based on

it, a prototype is made. Completely built sample model is shown to user and based on his feedback, the SRS (System Requirements Specifications) document is prepared.

Evolutionary prototyping

Evolutionary Prototyping (also known as breadboard prototyping) is quite different from Throwaway Prototyping. The main goal when using Evolutionary Prototyping is to build a very robust prototype in a structured manner and constantly refine it. The reason for this is that the Evolutionary prototype, when built, forms the heart of the new system, and the improvements and further requirements will be built. When developing a system using Evolutionary Prototyping, the system is continually refined and rebuilt. Evolutionary prototyping acknowledges that we do not understand all the requirements and builds only those that are well understood. This technique allows the development team to add features, or make changes that couldn't be conceived during the requirements and design phase.

Evolutionary Prototypes have an advantage over Throwaway Prototypes in that they are functional systems. Although they may not have all the features the users have planned, they may be used on an interim basis until the final system is delivered.
It is not unusual within a prototyping environment for the user to put an initial prototype to practical use while waiting for a more developed version…The user may decide that a 'flawed' system is better than no system at all. To minimize risk, the developer does not implement poorly understood features. The partial system is sent to customer sites. As users work with the system, they detect opportunities for new features and give requests for these features to developers. Developers then take these enhancement requests along with their own and use sound configuration-management practices to change the software-requirements specification, update the design, recode and retest.

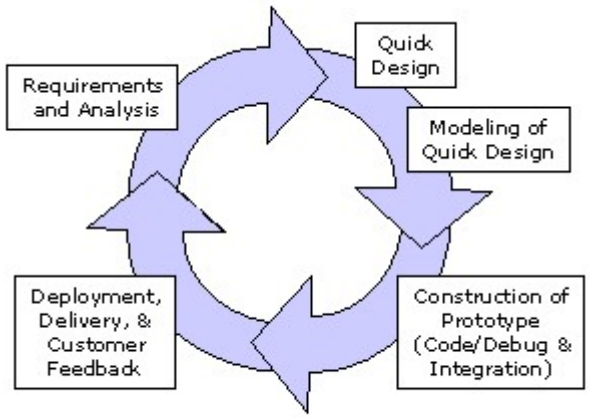

Fig.3.4 Prototype Model

Advantages of Prototyping Model

- When prototype is shown to the user, he gets a proper clarity and 'feel' of the functionality of the software and he can suggest changes and modifications and increasing user confidence.
- When client is not confident about the developer's capabilities, he asks for a small prototype to be built. Based on this model, he judges capabilities of developer.
- It reduces risk of failure, as potential risks can be identified early and mitigation steps can be taken.
- Iteration between development team and client provides a very good and conductive environment during project.
  Time required to complete the project after getting final the SRS reduces, since the developer has a better idea about how he should approach the project.

Disadvantages of Prototyping Model:

- Once we get proper requirements from client after showing prototype model, it may be of no use. That is why, sometimes we refer to the prototype as "Throw-away" prototype.
- It is a slow process.
- Too much involvement of client is not always preferred by the developer.
- Too many changes can disturb the rhythm of the development team.

3.4 Project Cost Estimation Model

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

  Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required are segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique

This technique assumes the software as a product of various compositions.

There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation. These formulae are based on LOC or FPs.

- **Putnam Model**

  This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

3.5 Project Scheduling

The project schedule is the tool that communicates what work needs to be performed, which resources of the organization will perform the work and the timeframes in which that work needs to be performed. The project schedule should reflect all of the work associated with delivering the project on time.

3.5.1 Pert Chart

A PERT chart is a project management tool used to schedule, organize, and coordinate tasks within a project. PERT stands for Program Evaluation Review Technique.

The PERT chart is sometimes preferred over the Gantt chart, another popular project management charting method, because it clearly illustrates task dependencies. On the other hand, the PERT chart can be much more difficult to interpret, especially on complex projects.

The proposed Pert chart is given as:

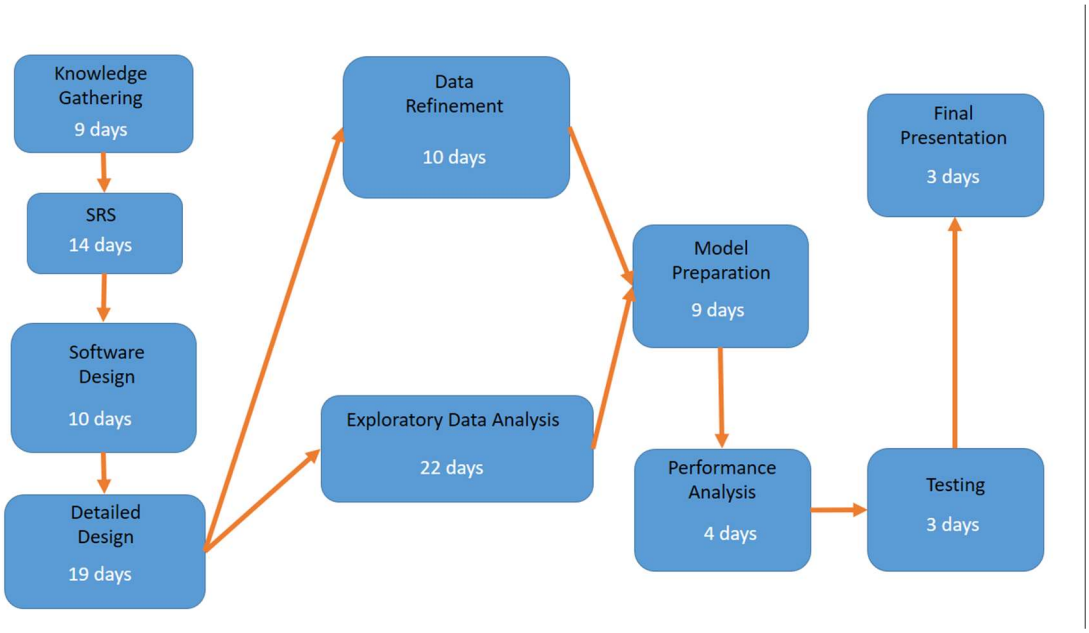

Fig 3.5 Pert diagram

3.5.2 GANTT Chart

The Gantt chart is the most widely used chart in project management. These charts are useful in planning a project and defining the sequence of tasks that require completion. In most instances, the chart displays as a horizontal bar chart. Horizontal bars of different lengths represent the project timeline that includes task sequences, duration, and start and end dates for each task. The horizontal bar also shows how much of a task requires completion. The horizontal bar length is proportional to the time necessary for a task's completion. In addition, the project tasks are visible on the vertical bar.

A Gantt chart aids in scheduling, managing, and monitoring specific tasks and resources in a project. The chart shows the project timeline that includes scheduled and completed work over a period. The Gantt chart aids project managers in communicating project status or plans and helps ensure the project remains on track.

The chart identifies tasks that may be executed in parallel and identifies tasks that cannot be started or finished until other tasks are complete. The Gantt chart aids in detecting potential bottlenecks and helps to identify tasks that may have been excluded from the project timeline.

The chart depicts task slack time or additional time for completion of a task that should not delay the project, non-critical activities that may be delayed and critical activities

that must executed on time. For example, if the project is installing new software on a server, the project tasks that require completion are conducting research, selecting a software, testing software and installing software. A milestone is selecting a software. These tasks appear as vertical bars on the chart. Each task takes 10 days to complete, and each task is dependent on the previous task. A critical activity is testing the software in the development and test environments. The task start and end dates, duration, and milestones appear as horizontal bars. The percentage complete for each task also displays on the horizontal bars. The project duration is 97 days. The project start date is 21 September 2017, and the project end date is 18 April 2018, which is based on workdays.

Task	Start Date	End Date	Duration
Knowledge Gathering	21-09-2017	30-09-2017	9
SRS	01-10-2017	15-10-2017	14
Software Design	16-10-2017	26-10-2017	10
Detailed Design	27-10-2017	15-11-2017	19
Data Refinement	20-03-2018	30-03-2018	10
Exploratory Data Analysis	15-03-2018	06-04-2018	22
Model Preparation	01-04-2018	10-04-2018	9
Performance Analysis	08-04-2018	12-04-2018	4
Testing	12-04-2018	15-04-2018	3
Final Presentation	15-04-2018	18-04-2018	3

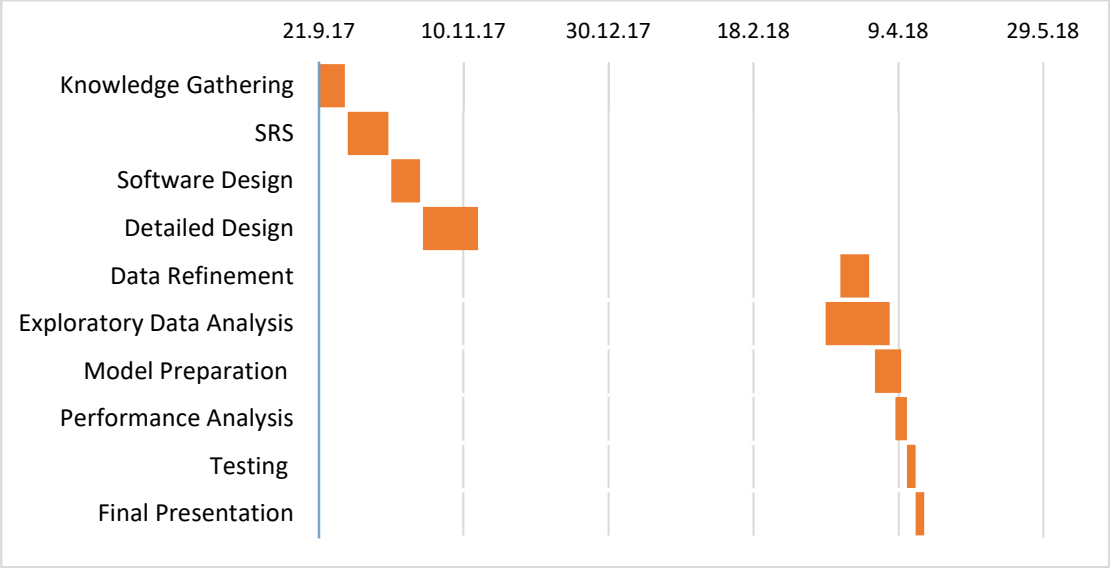

Fig 3.6 Gantt chart